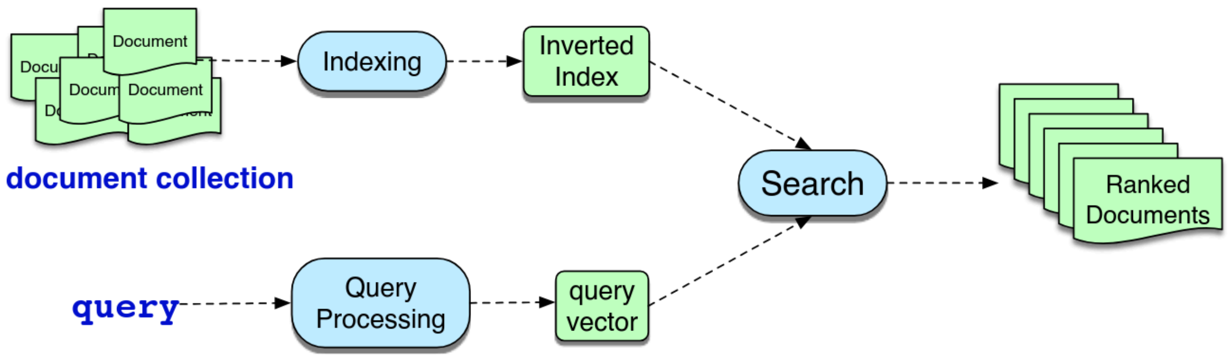
Natural Language Processing

Lab 11: Information Retrieval

Objective

This lab aims to familiarize you with information retrieval through term frequency-inverse document frequency (tf-idf). You'll explore how documents are searched and retrieved while using data structures specifically designed for tf-idf, which is foundational in natural language processing (NLP) for tasks like searching the web or fetching domain specific books in a digital library.

Theory



We don't use raw word counts in IR, instead computing a term weight for each document word. One such term weighting scheme is the term frequency-inverse document frequency (tf-idf).

The term frequency tells us how frequent the word is; words that occur more often in a document are likely to be informative about the document's contents. We usually use the $\log_{10}$ of the word frequency, rather than the raw count. The intuition is that a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document. Because we can't take the log of 0, we normally add 1 to the count.

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1)$$

The document frequency **df**$_t$ of a term **t** is the number of documents it occurs in. Terms that occur in only a few documents are useful for discriminating those documents from the rest of the collection; terms that occur across the entire collection aren't as helpful. The inverse document frequency or **idf** term weight is defined as:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

where **N** is the total number of documents in the collection, and **df$_t$** is the number of documents in which term **t** occurs. The fewer documents in which a term occurs, the higher this weight; the lowest weight of 0 is assigned to terms that occur in every document.

The **tf-idf** value for word **t** in document **d** is then the product of term frequency **tf$_{t,d}$** and **IDF**

$$\text{tf-idf}(t,d) = \mathrm{tf}_{t,d} \cdot \mathrm{idf}_t$$

We score document **d** by the cosine of its vector **d** with the query vector **q**

$$\mathrm{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{|\mathbf{q}||\mathbf{d}|}$$

Document Vectors Creation

Example document set:

```
Doc1: "The cat sat on the mat"
Doc2: "The dog chased the cat"
Doc3: "The bird flew away"
```

Perform the preprocessing steps for each document

- Tokenization: Split into individual words
  `Doc1: ["The", "cat", "sat", "on", "the", "mat"]`

- Lowercasing: Convert to lowercase
  `Doc1: ["the", "cat", "sat", "on", "the", "mat"]`

- Stop Word Removal: Remove all stop words
  `Doc1: ["cat", "sat", "mat"]`

- Lemmatization: Reduce to root word

```
     Doc1: ["cat", "sit", "mat"]
```

Collect all unique terms from all processed documents:

```
Vocab: ["cat", "sit", "mat", "dog", "chase", "bird", "fly", "away"]
  (Index: 0      1      2      3       4       5       6      7)
```

Calculate TF (Term Frequency) for each document

```
Doc1: ["cat", "sit", "mat"]
cat: 0.3010, sit: 0.3010, mat: 0.3010, others: 0

Doc2: ["dog", "chase", "cat"]
dog: 0.3010, chase: 0.3010, cat: 0.3010, others: 0

Doc3: ["bird", "fly", "away"]
bird: 0.3010, fly: 0.3010, away: 0.3010, others: 0
```

Calculate IDF

Term	**df(t)**	**IDF Calculation**	**IDF (log base 10)**
cat	2	log(3/2) = log(1.5)	0.1761
sit	1	log(3/1) = log(3)	0.4771
mat	1	log(3/1) = log(3)	0.4771
dog	1	log(3/1) = log(3)	0.4771
chase	1	log(3/1) = log(3)	0.4771
bird	1	log(3/1) = log(3)	0.4771
fly	1	log(3/1) = log(3)	0.4771
away	1	log(3/1) = log(3)	0.4771

Calculate TF-IDF for each document

For Doc1:

```
cat = 0.3010 * 0.1761 = 0.0530
sit = 0.3010 * 0.4771 = 0.1436
mat = 0.3010 * 0.4771 = 0.1436
dog = 0.0000 * 0.4771 = 0.0000
chase = 0.0000 * 0.4771 = 0.0000
bird = 0.0000 * 0.4771 = 0.0000
```

```
fly = 0.0000 * 0.4771 = 0.0000
away = 0.0000 * 0.4771 = 0.0000
Doc1 Vector = [0.0530, 0.1436, 0.1436, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000]
```

For Doc2:

```
cat = 0.3010 * 0.1761 = 0.0530
sit = 0.0000 * 0.4771 = 0.0000
mat = 0.0000 * 0.4771 = 0.0000
dog = 0.3010 * 0.4771 = 0.1436
chase = 0.3010 * 0.4771 = 0.1436
bird = 0.0000 * 0.4771 = 0.0000
fly = 0.0000 * 0.4771 = 0.0000
away = 0.0000 * 0.4771 = 0.0000
Doc2 Vector = [0.0530, 0.0000, 0.0000, 0.1436, 0.1436, 0.0000, 0.0000,
0.0000]
```

For Doc3:

```
cat = 0.0000 * 0.1761 = 0.0000
sit = 0.0000 * 0.4771 = 0.0000
mat = 0.0000 * 0.4771 = 0.0000
dog = 0.0000 * 0.4771 = 0.0000
chase = 0.0000 * 0.4771 = 0.0000
bird = 0.3010 * 0.4771 = 0.1436
fly = 0.3010 * 0.4771 = 0.1436
away = 0.3010 * 0.4771 = 0.1436
Doc3 Vector = [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1436, 0.1436,
0.1436]
```

Query Vector Creation

Example query:

```
Query: "cat and dog running"
```

Same preprocessing as documents:

- Tokenize: `["cat", "and", "dog", "running"]`
- Lowercase: `["cat", "and", "dog", "running"]`
- Remove stop words: `["cat", "dog", "running"]`
- Lemmatization: `["cat", "dog", "run"]`

Query vector MUST use the SAME vocabulary as documents! We will ignore "run" since it's not in our vocabulary.

```
Query: ["cat", "dog"]
```

Calculate TF (Term Frequency) for query

```
query_vec: ["cat", "dog"]
cat: 0.3010, dog: 0.3010, others: 0
```

Query uses the SAME IDF values calculated from the document corpus!

```
cat: 0.1761
dog: 0.4771
```

Calculate Query TF-IDF

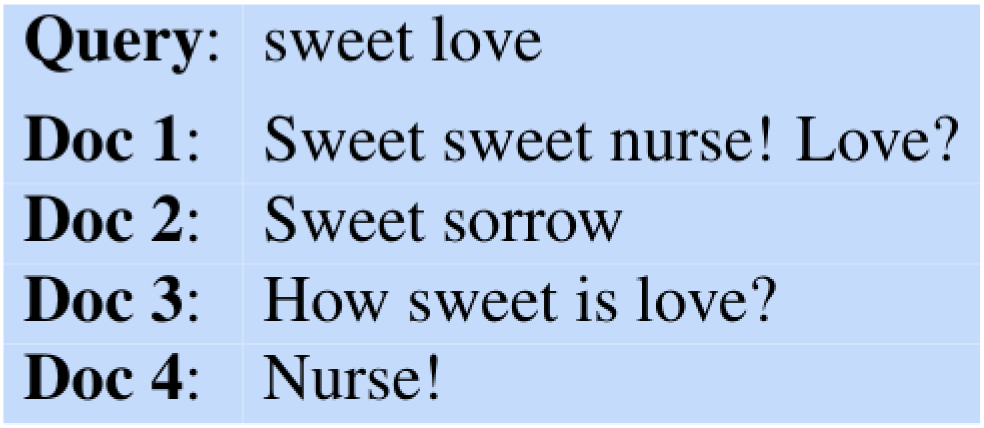
```
cat = 0.3010 * 0.1761 = 0.0530
sit = 0.0000 * 0.4771 = 0.0000
mat = 0.0000 * 0.4771 = 0.0000
dog = 0.3010 * 0.4771 = 0.1436
chase = 0.0000 * 0.4771 = 0.0000
bird = 0.0000 * 0.4771 = 0.0000
fly = 0.0000 * 0.4771 = 0.0000
away = 0.0000 * 0.4771 = 0.0000
Query Vector = [0.0530, 0.0000, 0.0000, 0.1436, 0.0000, 0.0000,
0.0000, 0.0000]
```

Inverted Index

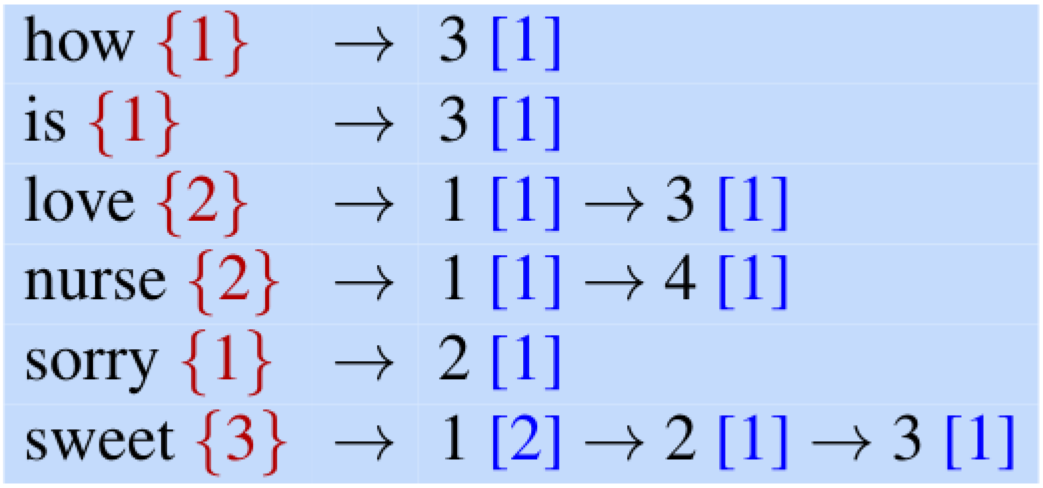
In order to compute scores, we need to efficiently find documents that contain words in the query. The basic search problem in IR is thus to find all documents **d** ∈ **C** that contain a term **q** ∈ **Q**.

The data structure for this task is the inverted index, which we use for making this search efficient, and also conveniently storing useful information like the document frequency and the count of each term in each document.

An inverted index, given a query term, gives a list of documents that contain the term. It consists of two parts, a dictionary and the postings. The dictionary is a list of terms (designed to be efficiently accessed), each pointing to a postings list for the term. A postings list is the list of document IDs associated with each term, which can also contain information like the term frequency or even the exact positions of terms in the document. The dictionary can also start the document frequency for each term. For example, a simple inverted index for our 4 sample documents above, with each word containing its document frequency in {}, and a pointer to a postings list that contains document IDs and term counts in [], might look like the following:



Query:	sweet love
Doc 1:	Sweet sweet nurse! Love?
Doc 2:	Sweet sorrow
Doc 3:	How sweet is love?
Doc 4:	Nurse!



how {1}	→	3 [1]
is {1}	→	3 [1]
love {2}	→	1 [1] → 3 [1]
nurse {2}	→	1 [1] → 4 [1]
sorry {1}	→	2 [1]
sweet {3}	→	1 [2] → 2 [1] → 3 [1]

Given a list of terms in the query, we can very efficiently get lists of all candidate documents, together with the information necessary to compute the tf-idf scores we need.

Task

Collect 6 blogs on different topics from the internet and store them as documents. Design an information retrieval system that gives the most similar document on a given query input. Use the inverted index for the data structure of the vocabulary.